

Property passed!
 $0.6688 N > 0 \text{ nrtr} < \text{MAX}$ $0.6688 N > 0 \text{ nrtr} < \text{MAX}$
PASS
 $0.6688 N > 0 \text{ nrtr} < \text{MAX}$ $0.6688 N > 0 \text{ nrtr} < \text{MAX}$
Property passed!

PASS Model Checker

Lijun Zhang

Joint work with Bjoern Wachter

Why PASS?

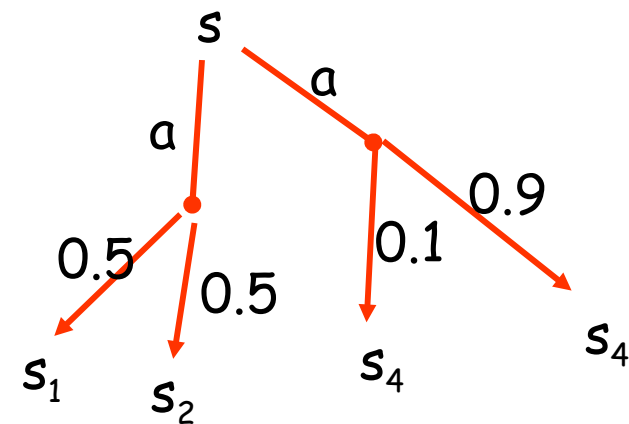
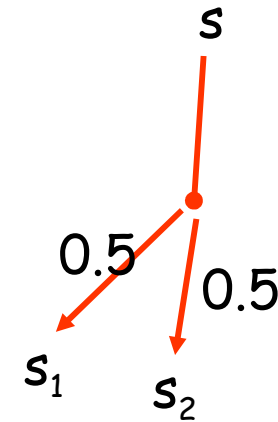
- Other model checkers
 - PRISM (Dave Parker et al)
 - MRMC (Ivan Zapreev et al)
 - ETMCC (Holger Hermanns et al)
 - Liquor (Frank Ciesinski et al)
 - Rapture (Pedro D'Argenio et al)
- What about infinite state systems?
 - Bisimulation does not work [DJ]
 - Abstraction for infinite CTMCs (QBDs) [DW, AR]
- PASS: Predicate Abstraction for Stochastic Systems
 - Abstraction interpretation [RW, JR] technique
 - High-level abstraction
 - Models (infinite MDPs, CTMCs, uCTMDPs)

Outline

- MDP and the logic PCTL
- Probabilistic Programs
- Predicate abstraction
- Experiments
- Ongoing works

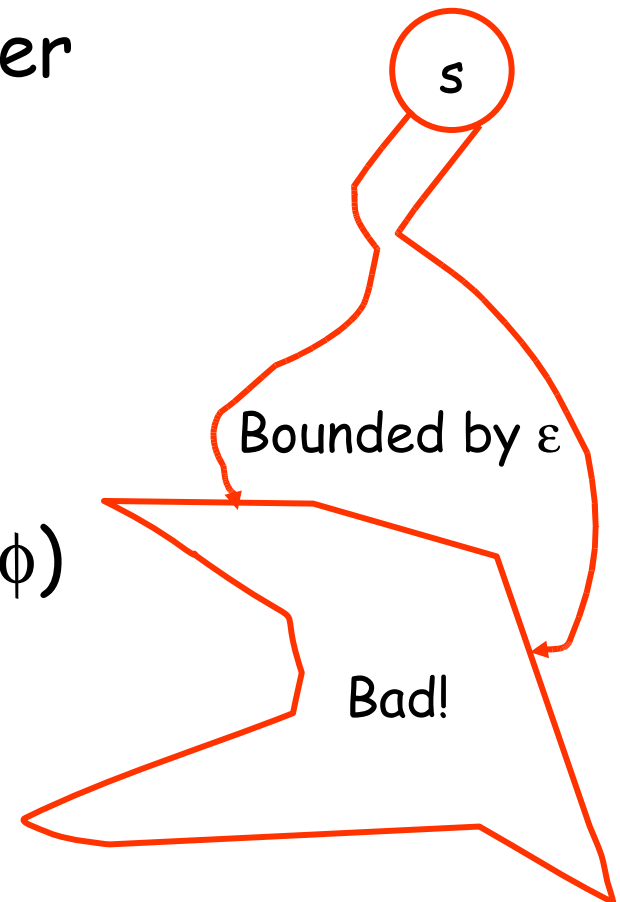
Recall the definitions

- DTMCs : Discrete-time Markov Chains
- MDPs: Markov decision processes
 - extension of DTMC with nondeterminism [SeLy95]
- Non-determinism in MDP can be resolved by schedulers
- Given an MDP M
 - a scheduler D induces a DTMC $D(M,A)$
 - there is a unique probabilistic measure $\Pr_{D(M,A)}$ for $D(M,A)$



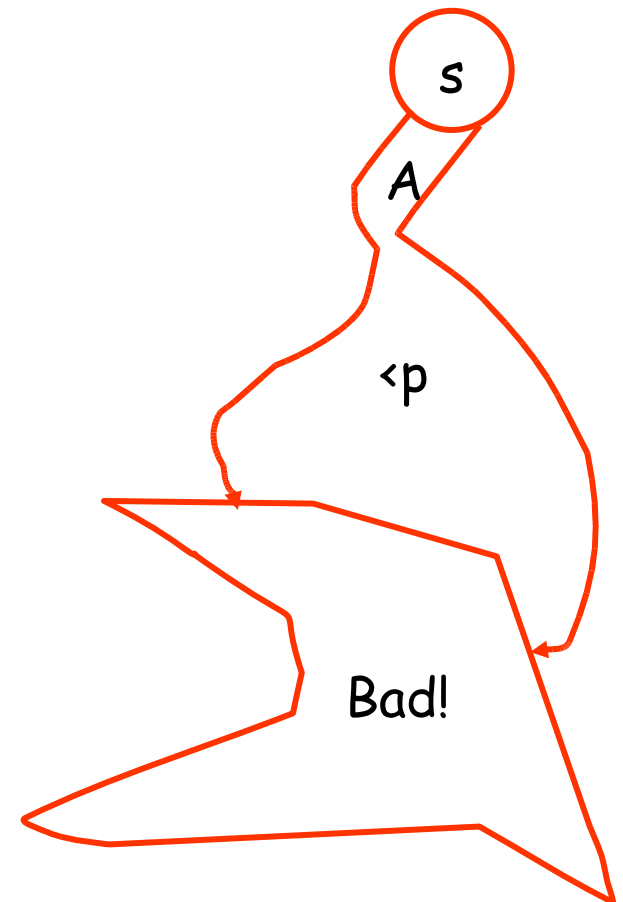
Recall the logic PCTL

- Extension of CTL with probabilistic operators [HJ]
- Logic to specify properties over probabilistic automata
- We consider only **safe** PCTL
- Syntax:
 - State formula
 - $\Phi = a \mid \neg \Phi \mid \Phi \wedge \Phi \mid P_{<p}(\phi)$
 - Path formula
 - $\phi = X \Phi \mid \Phi U \Phi$



Recall the logic PCTL

- Semantics of the probabilistic operator:
 - $s \models P_{<p}(\phi)$ iff $\Pr_{D(M,A)}\{\text{path satisfying } \phi\} < p$
for all schedulers A of M



Outline

- MDP and the logic PCTL
- Probabilistic Programs
- Predicate abstraction
- Experiments
- Ongoing works

PRISM input language

- Model type: DTMC, CTMC, MDP
 - CTMDP?
- Supports bounded integers, boolean data types
- The behaviour of each module is described by a set of guarded *commands*.
 - [action] guard \rightarrow $\text{prob}_1 : \text{update}_1 + \dots + \text{prob}_n : \text{update}_n;$

A PRISM example

mdp

module loop

const int N = 3;

bad:bool;

i:[0..N];

[a] !bad & i<N

-> 0.9:(i'=i+1)

+ 0.1:(bad'=(i=N-1))

endmodule

init !bad & i=0 endinit

A PRISM example

mdp

```
module loop
```

```
  const int N = 3;
```

```
  bad:bool;
```

```
  i:[0..N];
```

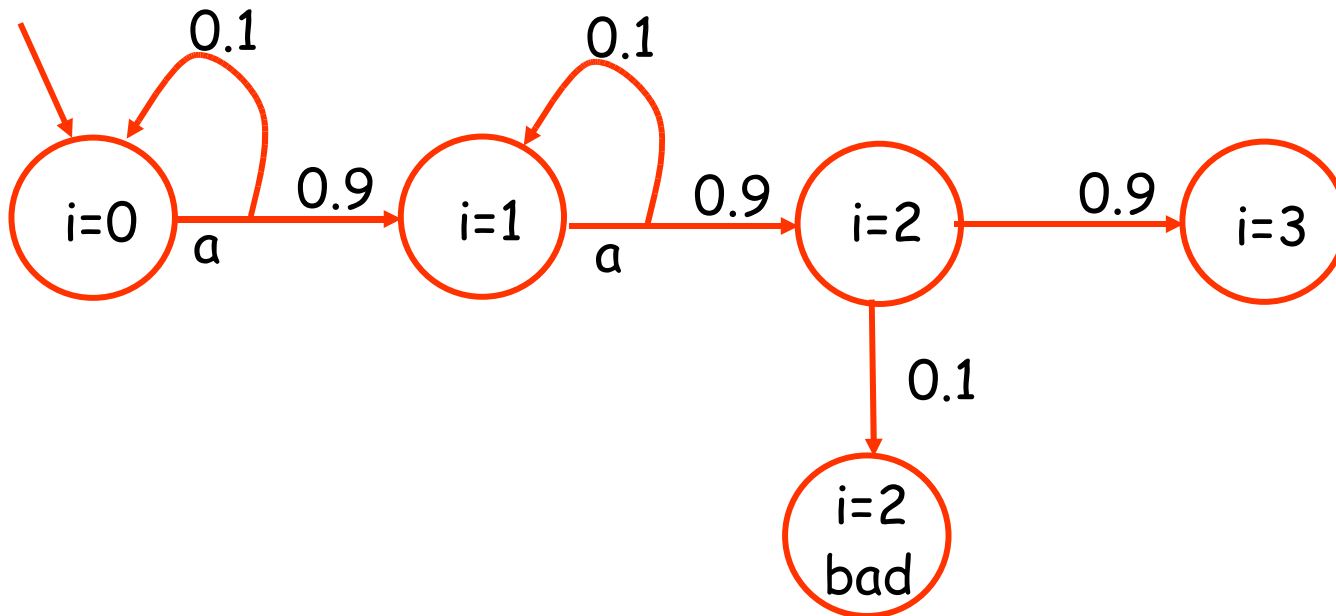
```
  [a] !bad & i<N
```

```
    -> 0.9:(i'=i+1)
```

```
    + 0.1:(bad'=(i=N-1))
```

```
endmodule
```

```
init !bad & i=0 endinit
```



A PRISM example

mdp

```
module loop
```

```
  const int N = 3;
```

```
  bad:bool;
```

```
  i:[0..N];
```

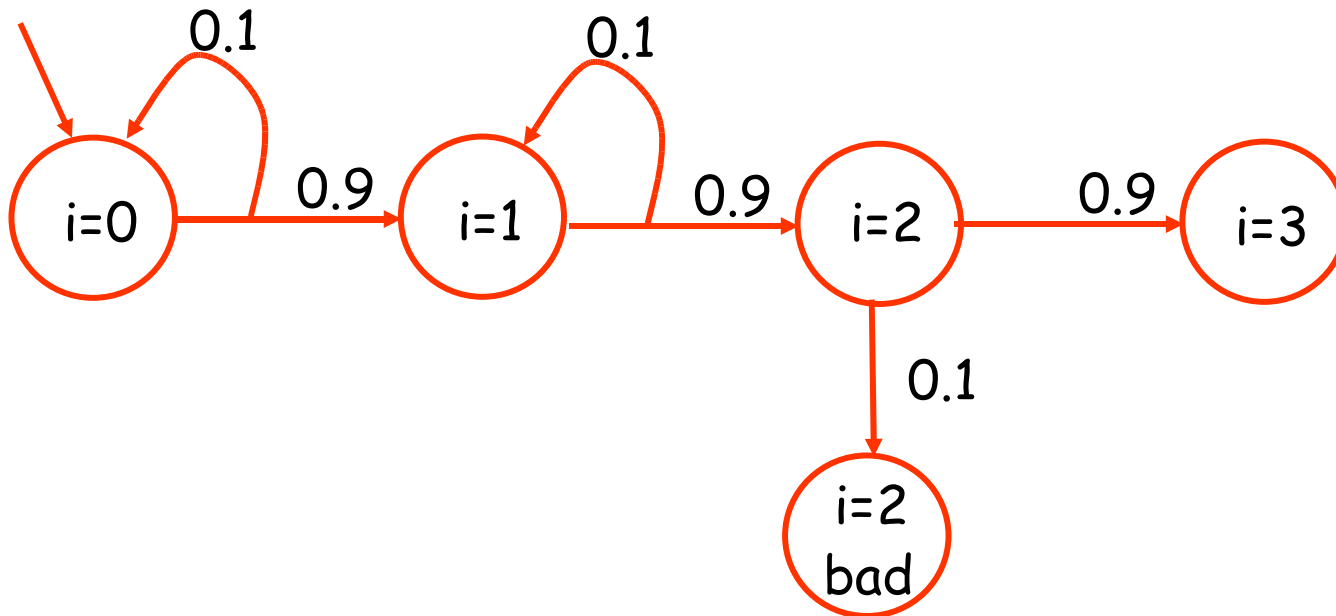
```
  [a] !bad & i<N
```

```
    -> 0.9:(i'=i+1)
```

```
      + 0.1:(bad'=(i=N-1))
```

```
endmodule
```

```
init !bad & i=0 endinit
```



Probabilistic programs

- Extension of the Prism input language
 - With extension of integer data type
 - $i : \text{int};$
 - With invariants:
 - $N : \text{int where } N > 5;$
 - Useful to verify properties for a family of MDPs
- We denote a probabilistic program by a tuple $PG = (X, \text{Init}, C)$

Probabilistic programs

mdp

```
module loop
```

```
N : int where N>2;
```

```
bad:bool;
```

```
i:int;
```

```
[a] !bad & i<N
```

```
-> 0.9:(i'=i+1)
```

```
+ 0.1:(bad'=(i=N-1))
```

```
endmodule
```

```
init !bad & i=0 endinit
```

mdp

```
module loop
```

```
const int N = 3;
```

```
bad:bool;
```

```
i:[0..N];
```

```
[a] !bad & i<N
```

```
-> 0.9:(i'=i+1)
```

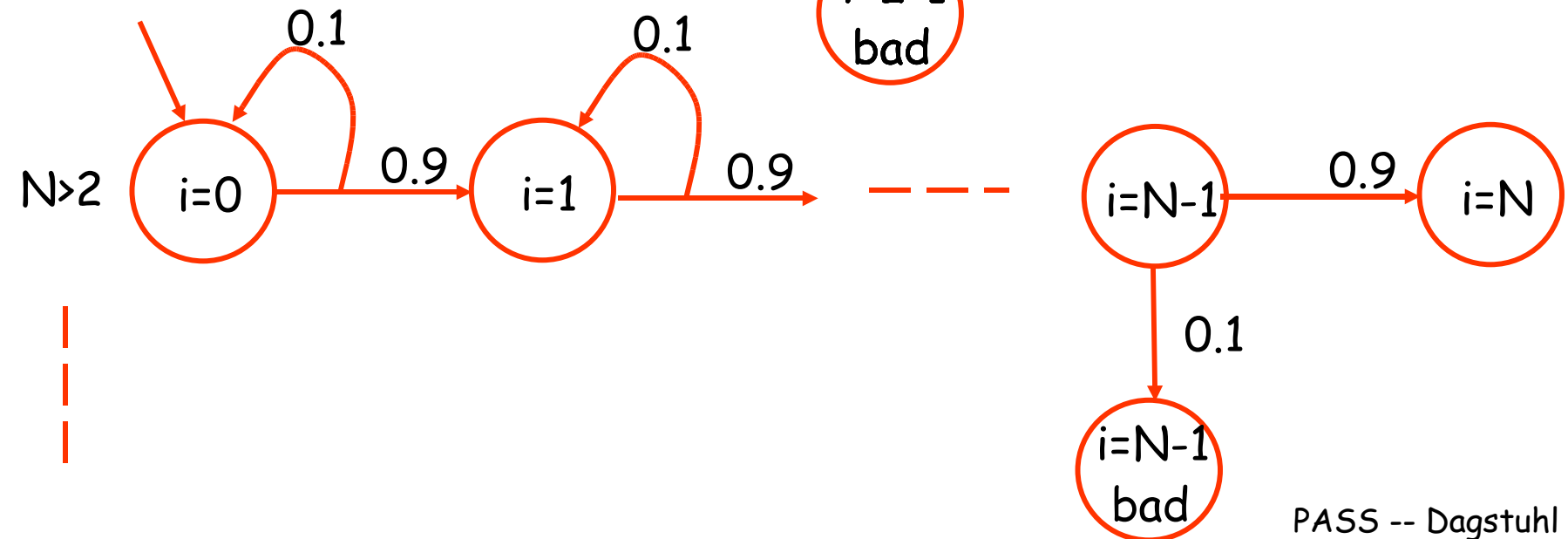
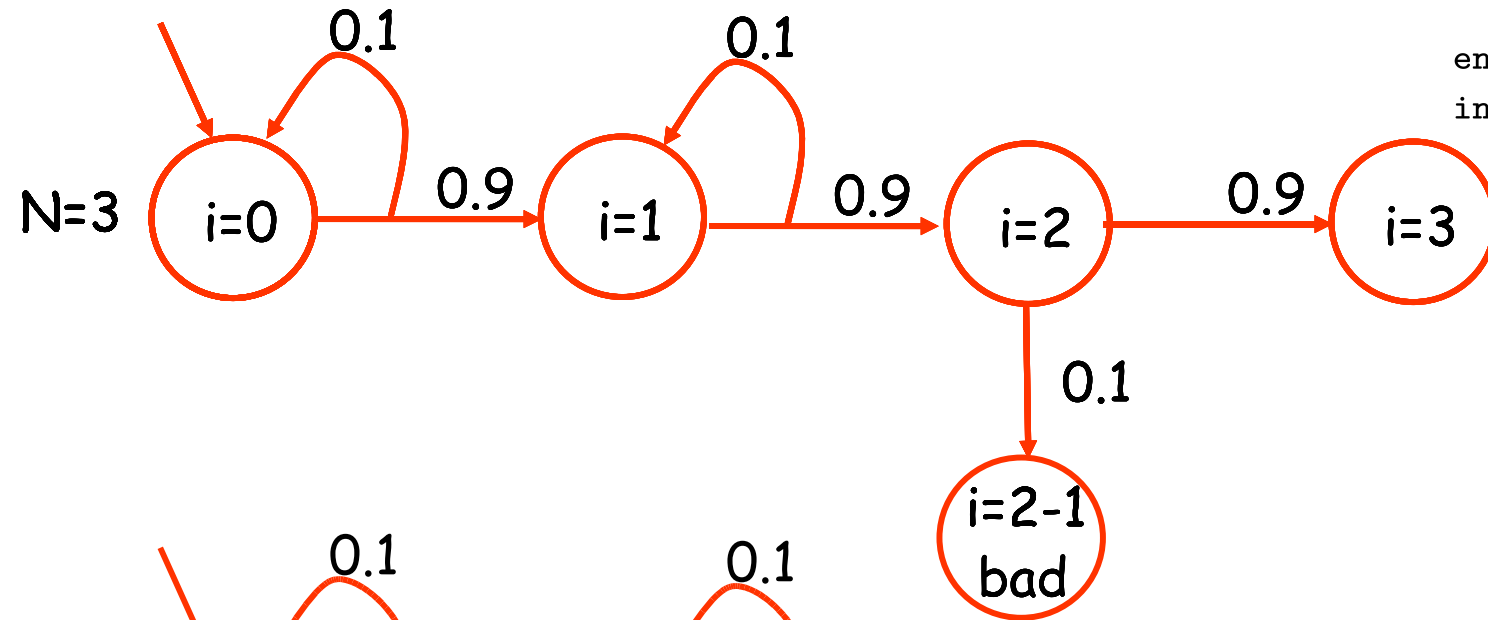
```
+ 0.1:(bad'=(i=N-1))
```

```
endmodule
```

```
init !bad & i=0 endinit
```

Probabilistic programs

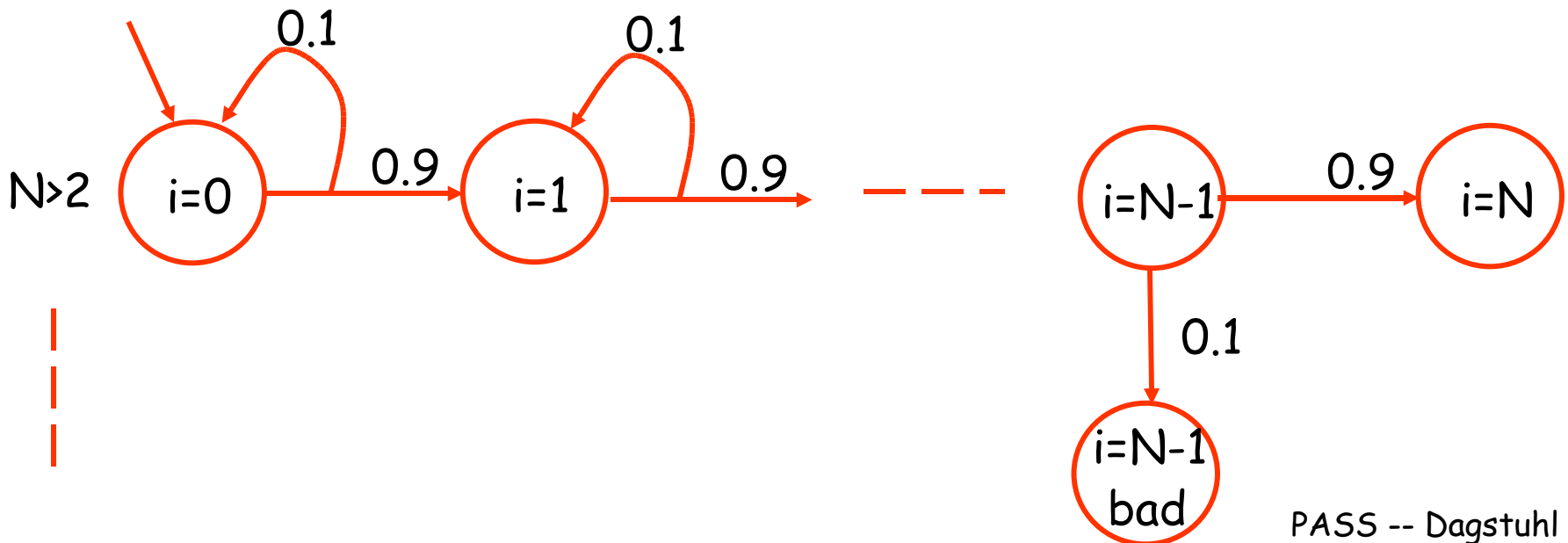
```
module loop
  N : int where N>2;
  bad:bool;
  i:int;
  [a] !bad & i<N
  -> 0.9:(i'=i+1)
  + 0.1:(bad'=(i=N-1))
endmodule
init !bad & i=0 endinit
```



Probabilistic programs

mdp

```
module loop
  N : int where N>2;
  bad:bool;
  i:int;
  [a] !bad & i<N
    -> 0.9:(i'=i+1)
    + 0.1:(bad'=(i=N-1))
endmodule
init !bad & i=0 endinit
```



Program semantics

□ Consider probabilistic program $PG=(X, \text{Init}, C)$

□ For a set Σ

- $\text{Distr}(\Sigma)$: the set of distributions

□ The semantics is a MDP $M = (\Sigma(X), I, \text{Act}, R)$

- The set of states $\Sigma(X)$

- Type consistent mapping of X to its domains

- I : set of states satisfying Init

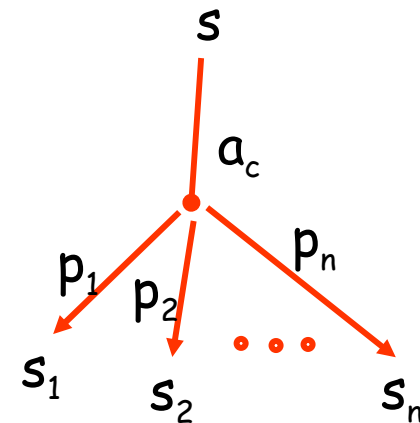
- $\text{Act} : \{a_c \mid c \text{ is a guarded command}\}$

- Semantics of the guarded command:

$[c] g \rightarrow p_1 : \text{update}_1 + \dots + p_n : \text{update}_n;$

- $\{ (s, a_c, \mu) \mid s \models g \text{ and } \mu(s') = \sum \{p_i \mid \text{update}_i \text{ lead to state } s'\} \}$

- R : disjunction of the semantics of all guarded commands in C



Outline

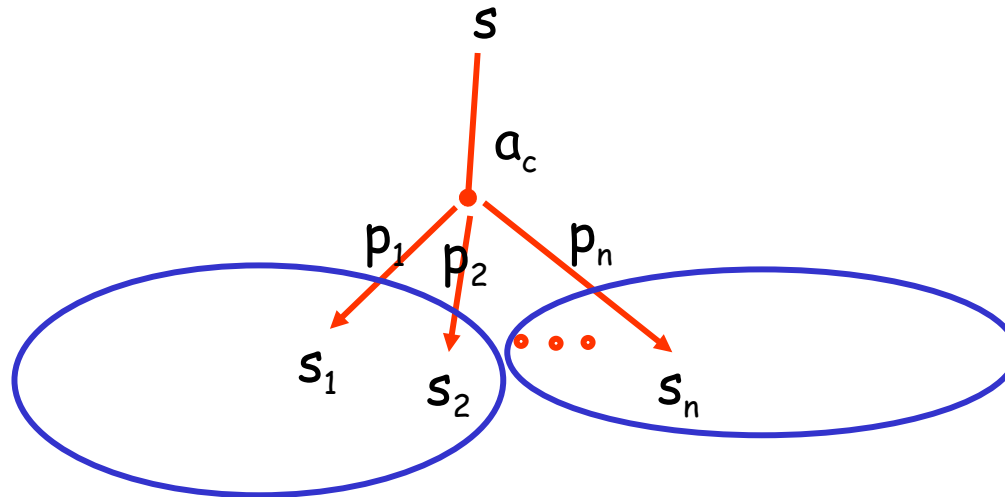
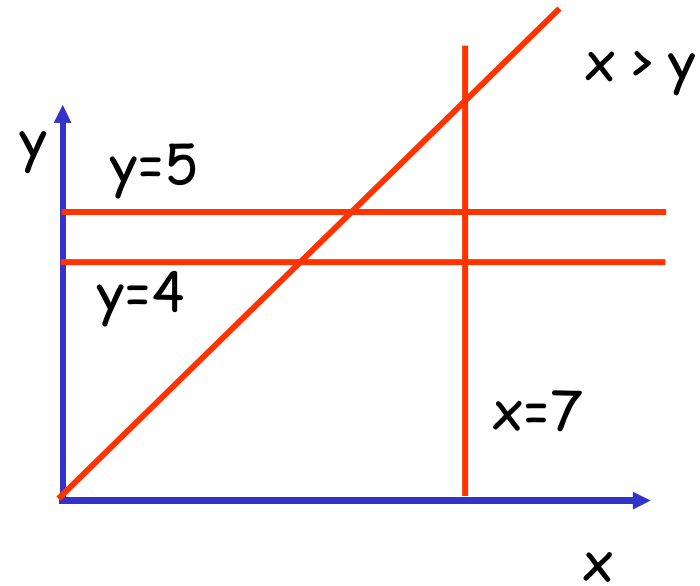
- MDP and the logic PCTL
- Probabilistic Programs
- Predicate abstraction
- Experiments
- Ongoing works

Predicate abstraction

- Predicates are boolean expressions over the program variables
 - $i=3, i > 5, i < N-1$
- For a probabilistic program PG , we consider a set of predicates $\Gamma = \{\gamma_1, \dots, \gamma_n\}$
 - Initially they consist of boolean expressions
 - in the guarded commands
 - in the initial condition
 - property to be checked
- Γ induces an equivalence relation \sim_Γ on Σ
 - $s \sim_\Gamma s'$ iff they satisfy the same predicates in Γ

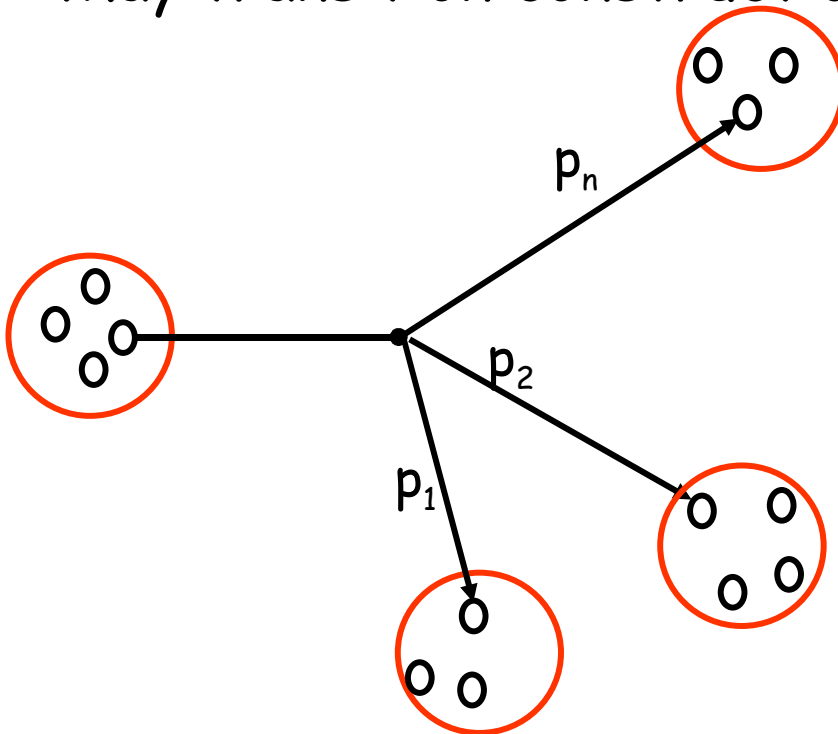
Abstract function

- Let Σ^* denote the set Σ / \sim_{Γ}
 - The set of **abstract states**
 - With abstract function $h : \Sigma \rightarrow \Sigma^*$
 - $h(s) = [s]_{\sim_{\Gamma}}$
 - For a distribution $\mu \in \text{Distr}(\Sigma)$, define $h(\mu) \in \text{Distr}(\Sigma^*)$



Quotient automata

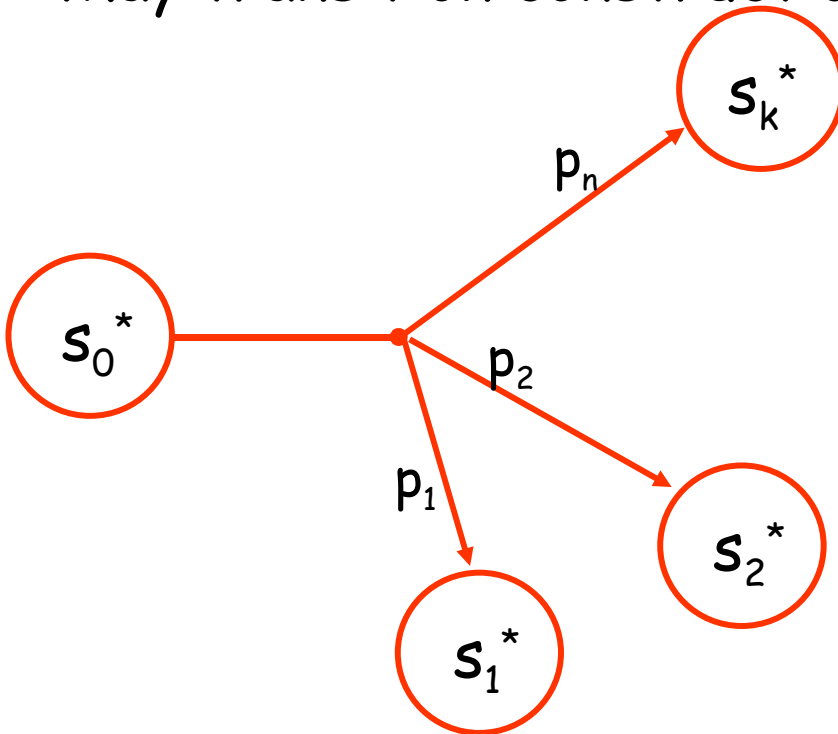
- For MDP M and $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ we define the quotient automata $M^* = (\Sigma^*, I^*, \text{Act}, R^*)$
 - $\Sigma^* = \Sigma / \sim_\Gamma$
 - $I^* = \{h(s) \mid s \in I\}$
 - $R^* = \{ (h(s), a, h(\mu)) \mid (s, a, \mu) \in R \}$
- May transition construction



\circ : a concrete state

Quotient automata

- For MDP M and $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ we define the quotient automata $M^* = (\Sigma^*, I^*, \text{Act}, R^*)$
 - $\Sigma^* = \Sigma / \sim_\Gamma$
 - $I^* = \{h(s) \mid s \in I\}$
 - $R^* = \{ (h(s), a, h(\mu)) \mid (s, a, \mu) \in R \}$
- May transition construction



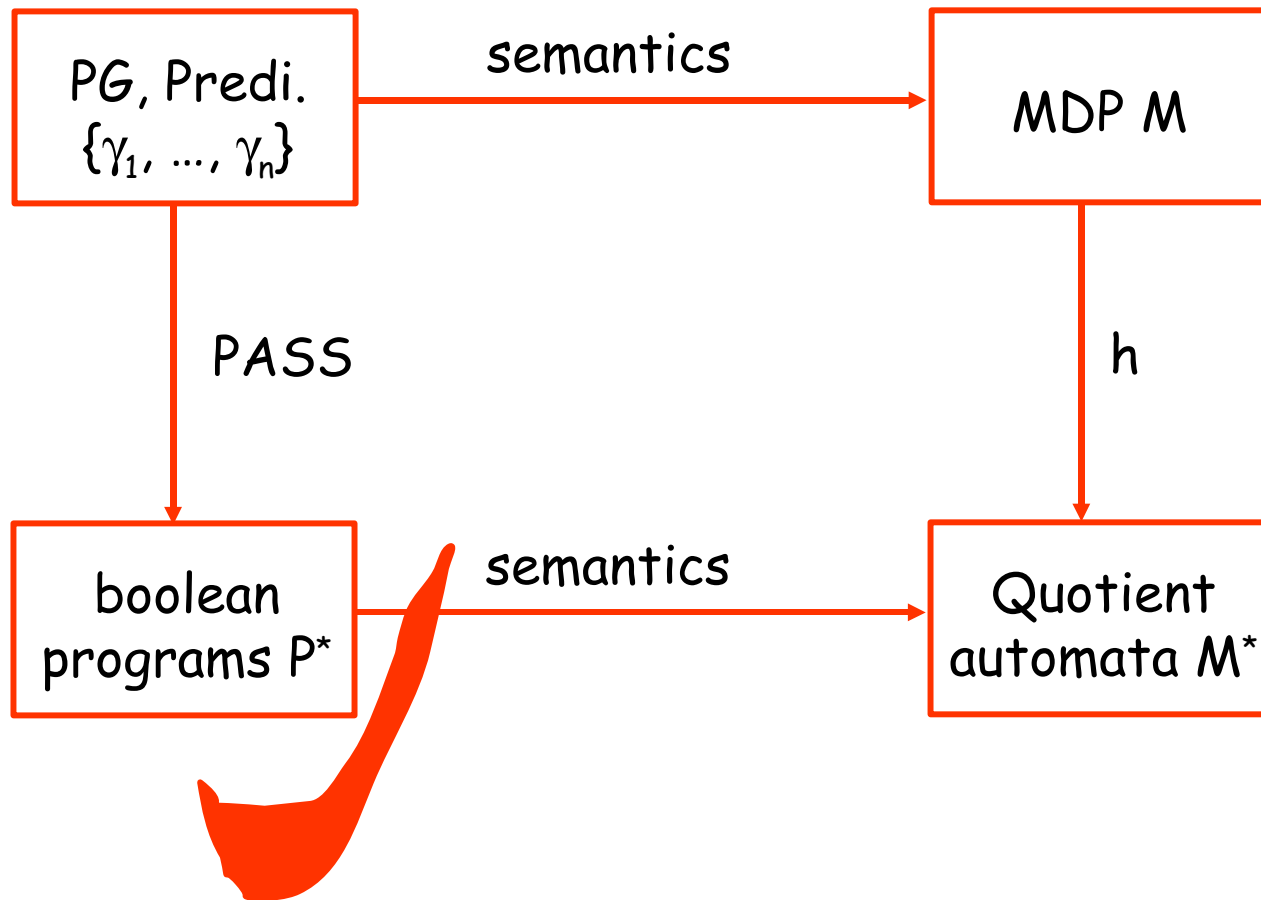
Quotient automata

Lemma: The quotient automata preserves safe PCTL formulas

Proof Idea

- The quotient automata M^* overapproximate the transitions.
- Hence, the probability of satisfying a path formula becomes higher in M^* .

Predicate abstraction



Boolean program

- Given $PG = (X, \text{Init}, C)$, predicates $\Gamma = \{\gamma_1, \dots, \gamma_n\}$
- Introduce boolean variables $B = \{b_1, \dots, b_n\}$ where b_i corresponds to the predicate γ_i
- The concretisation function [RW]
 - $\text{Con}(e) = e[b_i/\gamma_i]$
- How to compute the desired boolean program $P^* = (B, I^*, C^*)$
 - I^* : replace the expressions in Init by the corresponding boolean variables in B
- How to compute the abstract guarded commands C^* ?

Boolean program

- Consider the guarded command:
 $[c] g \rightarrow p_1 : \text{update}_1 + \dots + p_n : \text{update}_n;$
- $WP_{\text{update}_i}(e)$: the weakest precondition of e [TH]
- Consider the abstract states
 $s_0^* = (b_1^0, \dots, b_n^0), \dots, s_k^* = (b_k^0, \dots, b_k^n)$

s_k^*

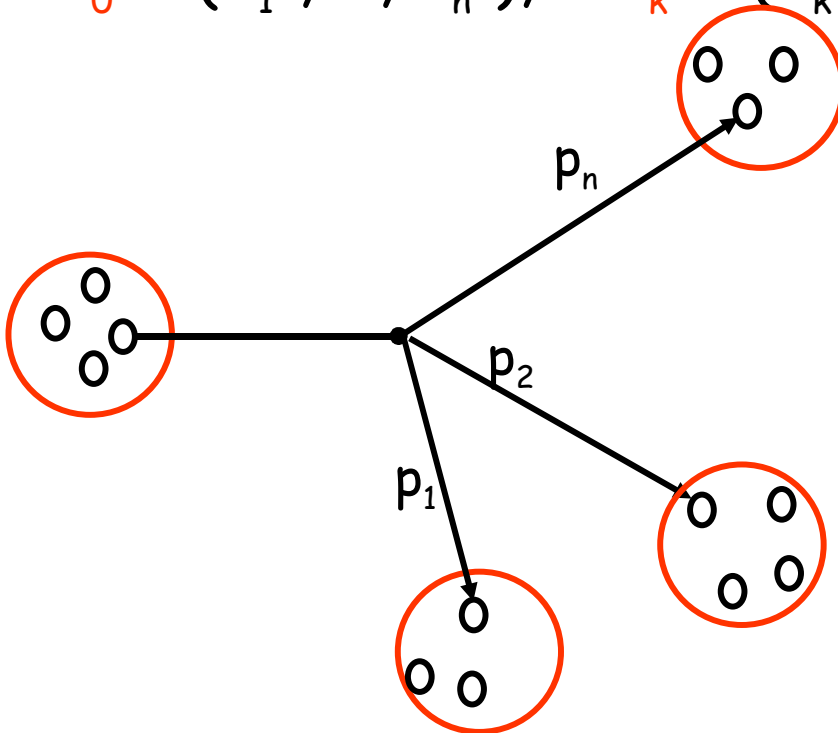
s_0^*

s_1^*

s_2^*

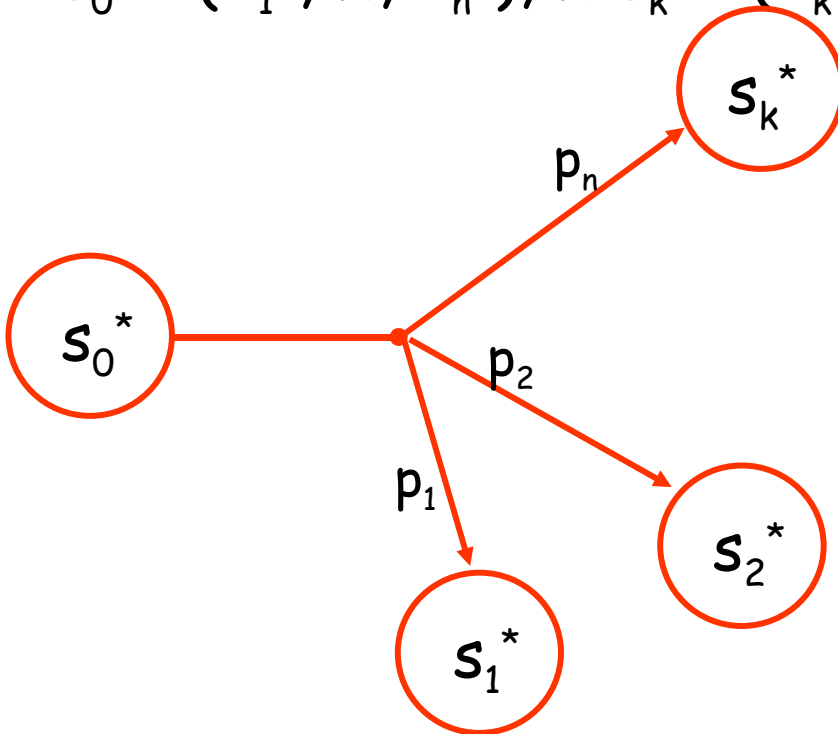
Boolean program

- Consider the guarded command:
 $[c] g \rightarrow p_1 : \text{update}_1 + \dots + p_n : \text{update}_n;$
- $WP_{\text{update}_i}(e)$: the weakest precondition of e [TH]
- Consider the abstract states
 $s_0^* = (b_1^0, \dots, b_n^0), \dots s_k^* = (b_k^0, \dots, b_k^n)$



Boolean program

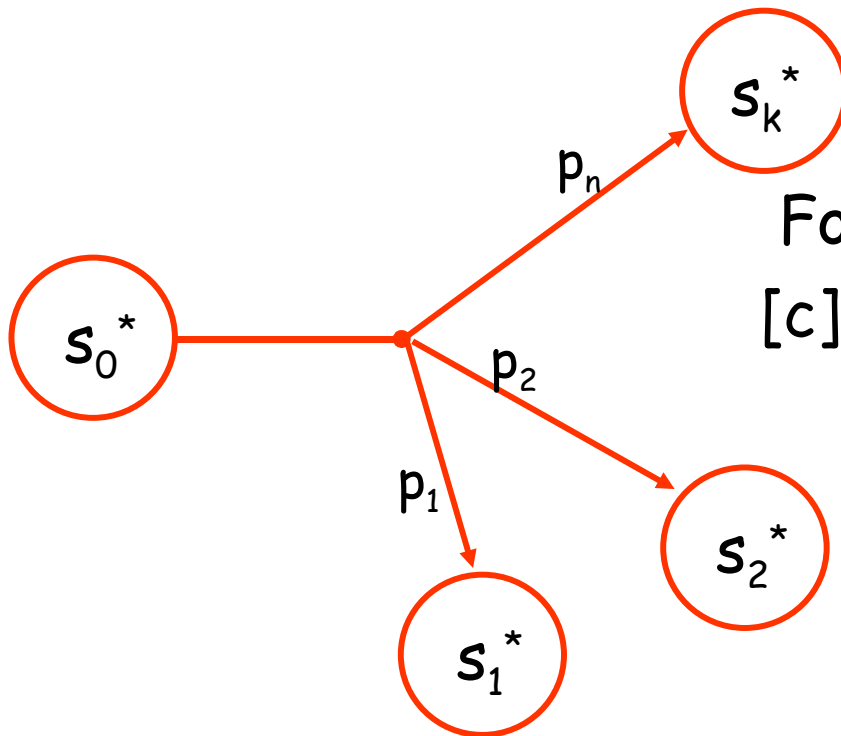
- Consider the guarded command:
 $[c] g \rightarrow p_1 : \text{update}_1 + \dots + p_n : \text{update}_n;$
- $WP_{\text{update}_i}(e)$: the weakest precondition of e [TH]
- Consider the abstract states
 $s_0^* = (b_1^0, \dots, b_n^0), \dots s_k^* = (b_k^0, \dots, b_k^n)$



Boolean program

- $s_0^* = (b_1^0, \dots, b_n^0), \dots, s_k^* = (b_k^0, \dots, b_k^n)$
- Consider the expression:

$$(SMT) \quad g \wedge \bigwedge_{j=1..n} [b_j^0 \Leftrightarrow \text{Con}(b_j^0)] \\ \wedge \bigwedge_{i=1..k} \bigwedge_{j=1..n} [b_j^i \Leftrightarrow \text{WP}_{\text{update}_i}(\text{Con}(b_j^i))]$$



For every solution introduce:
[c] $s_0^* \rightarrow p_1 : B' = s_1^* + \dots + p_n : B' = s_n^*$;

Boolean program

- How to get all of the solutions of

$$(SMT) \quad g \wedge \bigwedge_{j=1..n} [b_j^0 \Leftrightarrow \text{Con}(b_j^0)] \\ \wedge \bigwedge_{i=1..k} \bigwedge_{j=1..n} [b_j^i \Leftrightarrow \text{WP}_{\text{update}_i}(\text{Con}(b_j^i))]$$

- USE SMT-Solver!
- We used **Yices** (won SMT competition 2006)
[Dutertre, de Moura]
- Optimisations:
 - Decomposition
 - Relevant Predicates
 - Disjoint predicates
 - $\{i=1, i=2, i=3, \dots, i=666, i>666\}$

Probabilistic programs

- Consider the predicates:
 - $i < N-1$, $i = N-1$, bad

mdp

```
module loop
```

```
  N : int where N > 2;
```

```
  bad:bool;
```

```
  i:int;
```

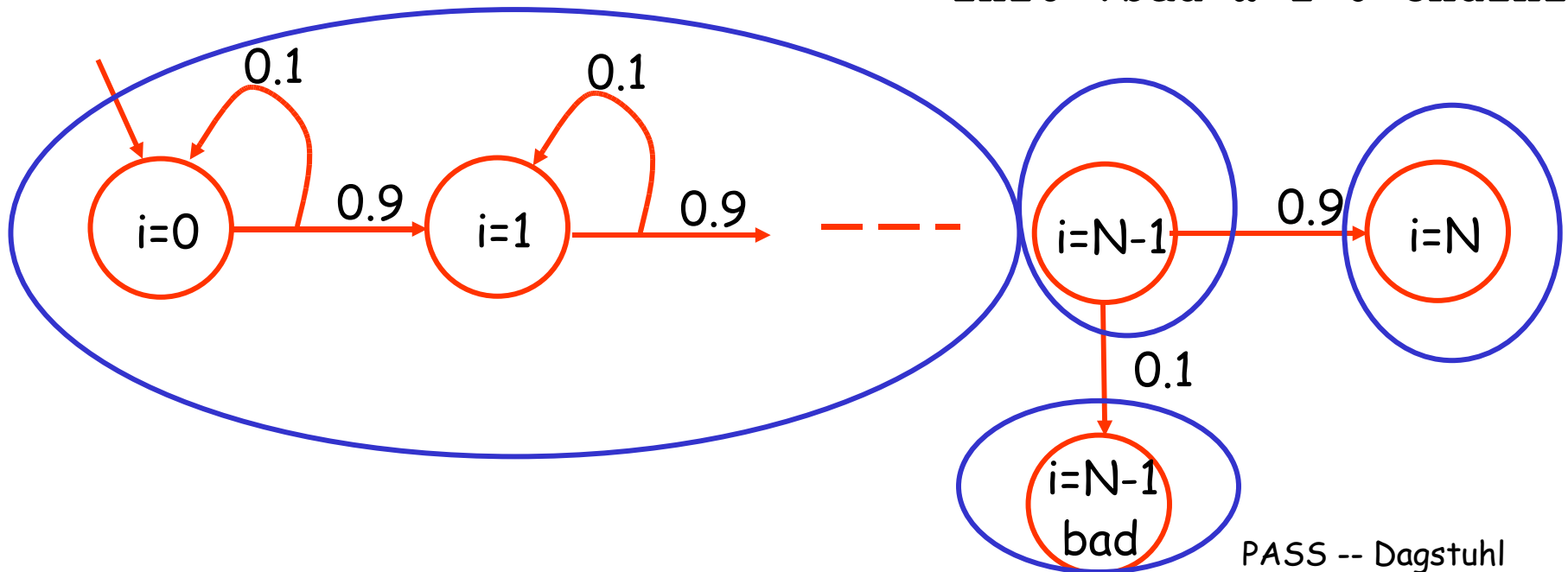
```
  [a] !bad & i < N
```

```
    -> 0.9:(i'=i+1)
```

```
    + 0.1:(bad'=(i=N-1))
```

```
endmodule
```

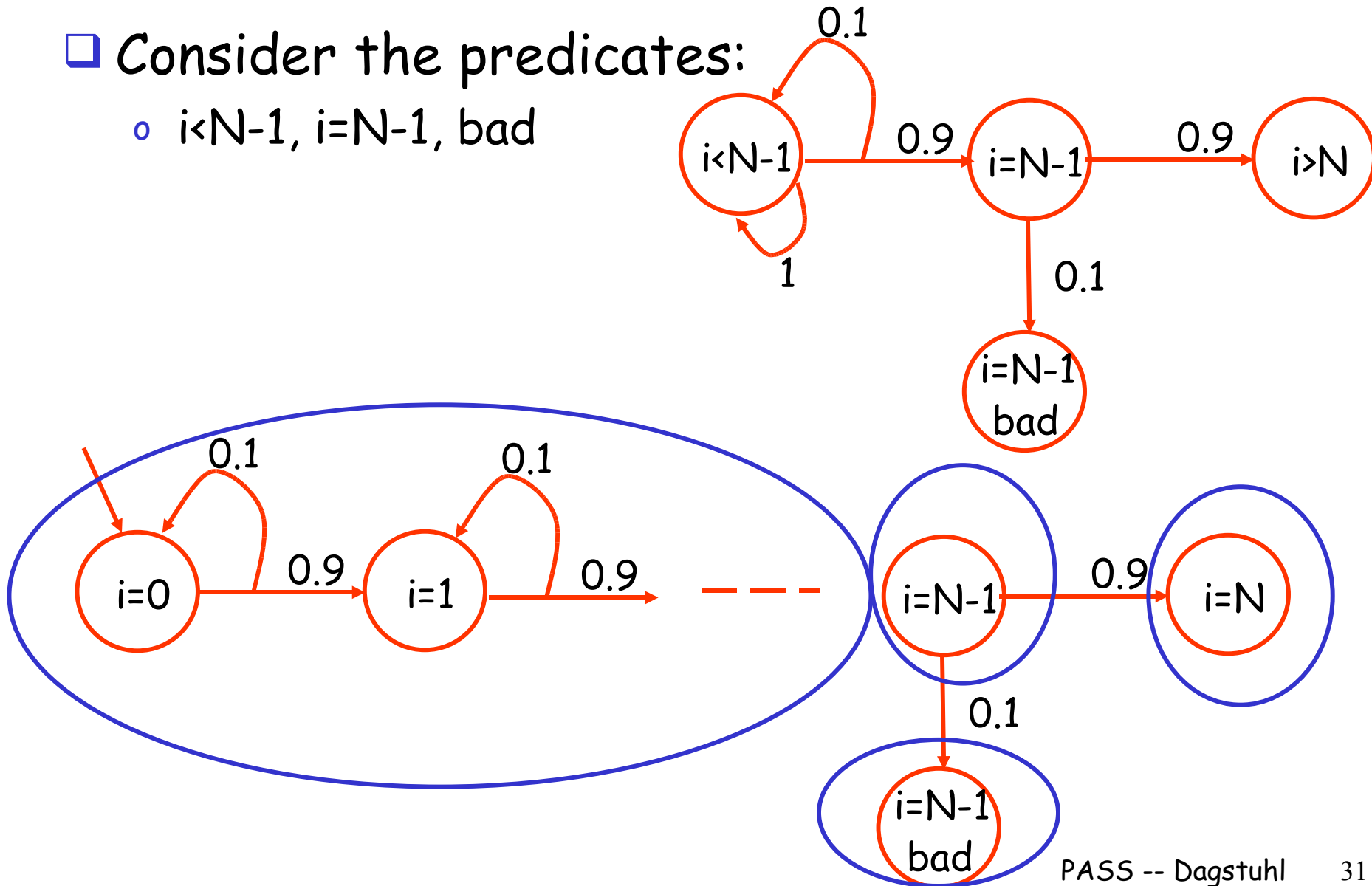
```
init !bad & i=0 endinit
```



Probabilistic programs

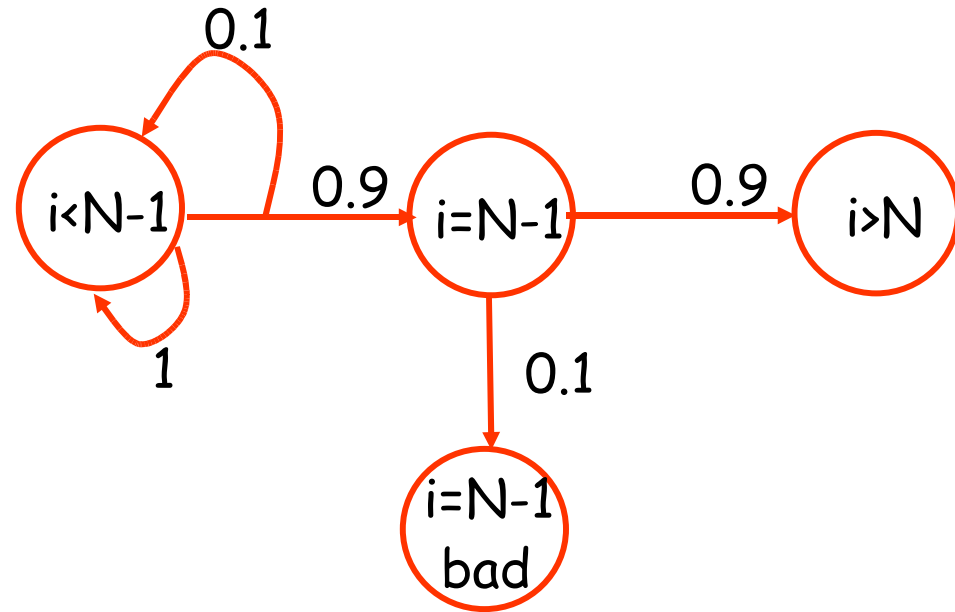
□ Consider the predicates:

- $i < N-1$, $i = N-1$, bad



Probabilistic programs

- Consider the predicates:
 - $i < N-1$, $i = N-1$, bad
- The same abstraction for all possible values of $N > 2!$



mdp

```
module loop
```

```
  const int N = 3;
```

```
  bad:bool;
```

```
  i:[0..N];
```

```
  [a] !bad & i < N
```

```
    -> 0.9:(i'=i+1)
```

```
    + 0.1:(bad'=(i=N-1))
```

```
endmodule
```

```
init !bad & i=0 endinit
```

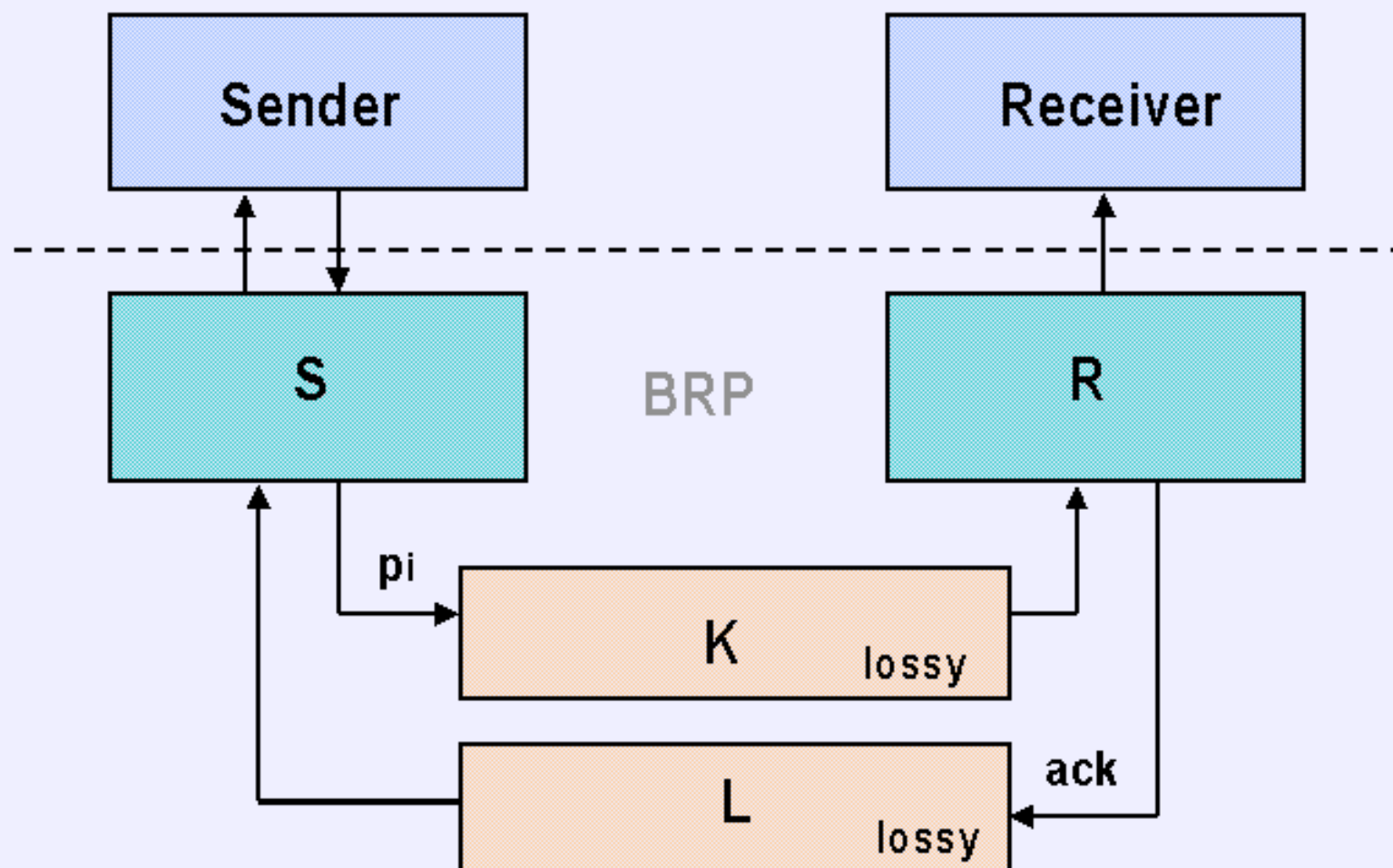
Outline

- MDP and the logic PCTL
- Probabilistic Programs
- Predicate abstraction
- Experiments
- Ongoing works

Overview of BRP

Input: file = p_1, \dots, p_n

Output: p_1, \dots, p_n



Experiments: BRP

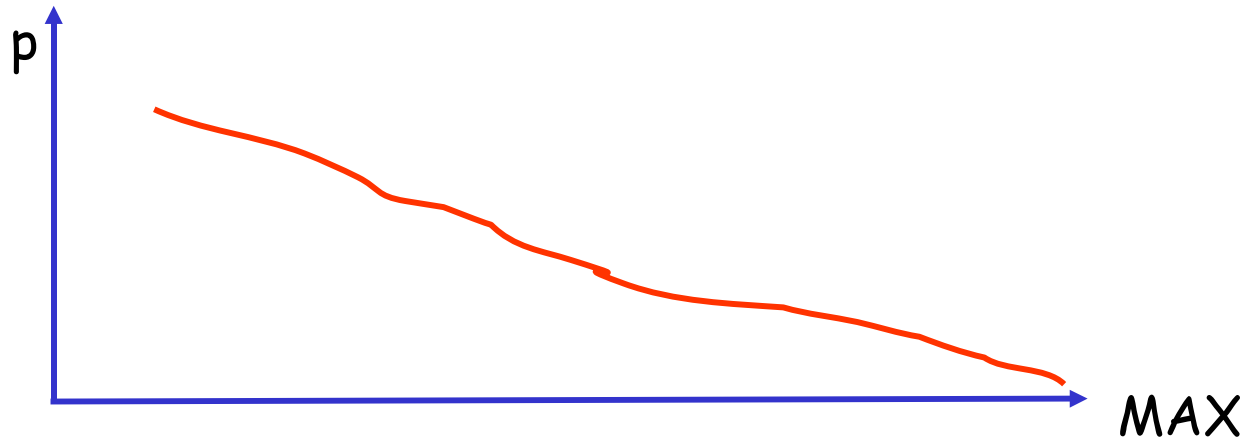
Consider the properties:

- ❑ A : sender reports an unsuccessful transmission but receiver got complete file
- ❑ B : sender reports a successful transmission but receiver didn't get complete file
- ❑ 1 : sender doesn't report a successful transmission
- ❑ 2 : sender reports an uncertainty on the success of the transmission
- ❑ 3 : sender reports an unsuccessful transmission after transmitting more than 8 chunks
- ❑ 4 : receiver doesn't receive any chunk of a file

Experiments: BRP

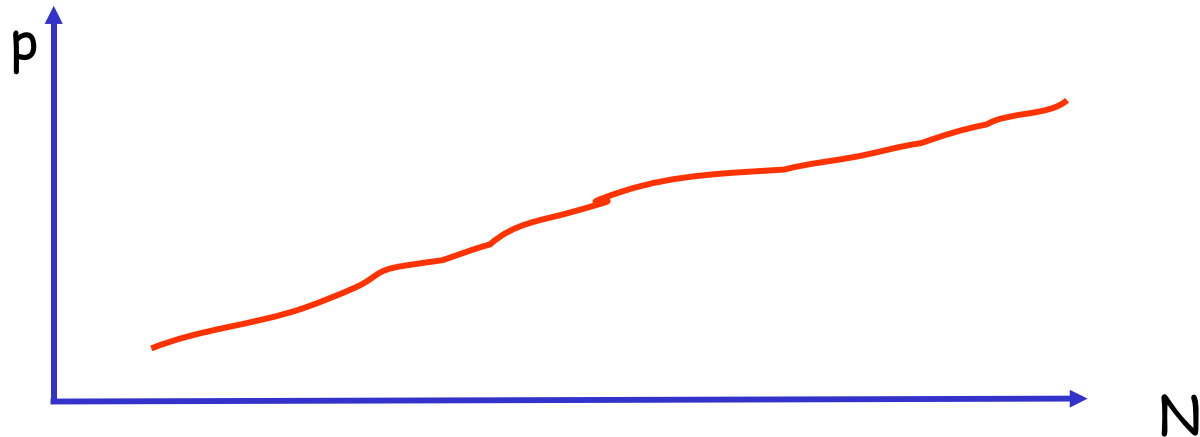
- Property A and B are invariant properties
 - // maximum number of retransmissions
MAX : int where MAX > 0;
 - // file length
N : int where invar N > 0;
- PASS construct the abstract model in 0.3 s
- The probability for A or B to hold is indeed 0
- We verify it once for all MAX>0 and N>0!

Experiments: BRP



| Properties 2 & 4, $P[N > 0, MAX \geq k]$ | | | | |
|--|---------|-------|------------|------------|
| k | $Preds$ | PASS | Property 2 | Property 4 |
| 2 | 53 | 642ms | 2.65E-5 | 8.00E-6 |
| 3 | 54 | 670ms | 7.89E-7 | 1.60E-7 |
| 4 | 55 | 712ms | 2.35E-8 | 3.20E-9 |
| 5 | 56 | 702ms | 7.00E-10 | 6.40E-11 |

Experiments: BRP

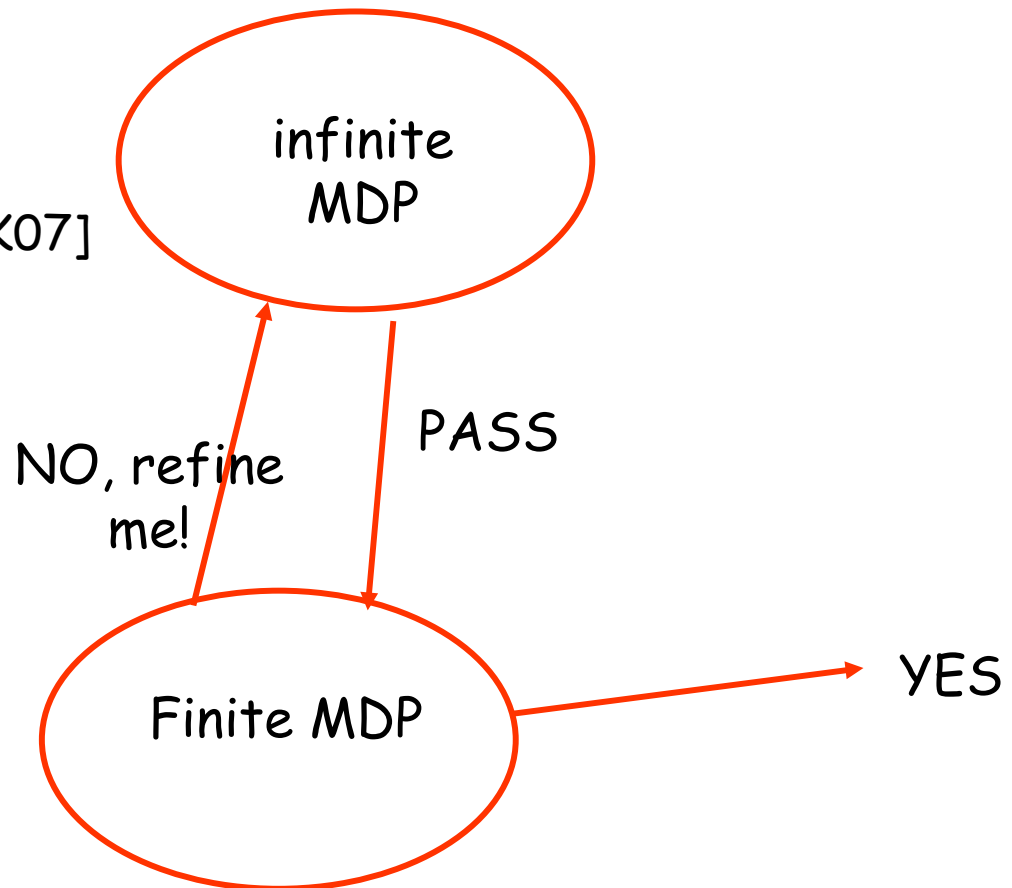


| Properties 1 & 3, $P[N = 16, MAX \geq k]$ | | | | |
|---|----|-------|---------|---------|
| 2 | 60 | 887ms | 4.23E-4 | 1.85E-4 |
| 3 | 61 | 900ms | 1.26E-5 | 5.25E-6 |
| 4 | 62 | 907ms | 3.76E-7 | 1.65E-7 |
| 5 | 63 | 940ms | 1.12E-8 | 4.90E-9 |

Ongoing works (I)

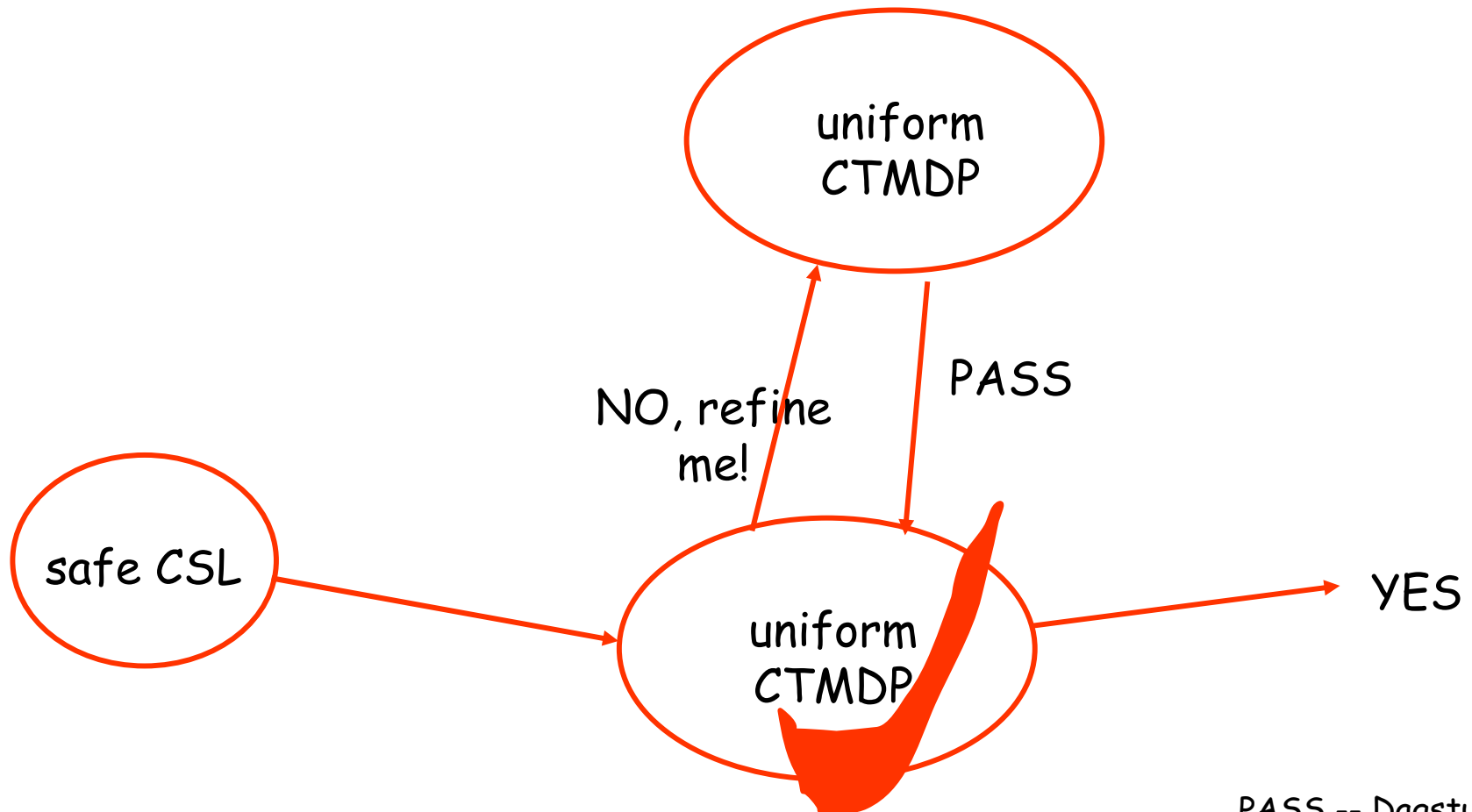
Automated refinement

- How can we refine the models?
- Generate additional predicates:
 - use counter-adversaries
 - use counter-examples [HK07]



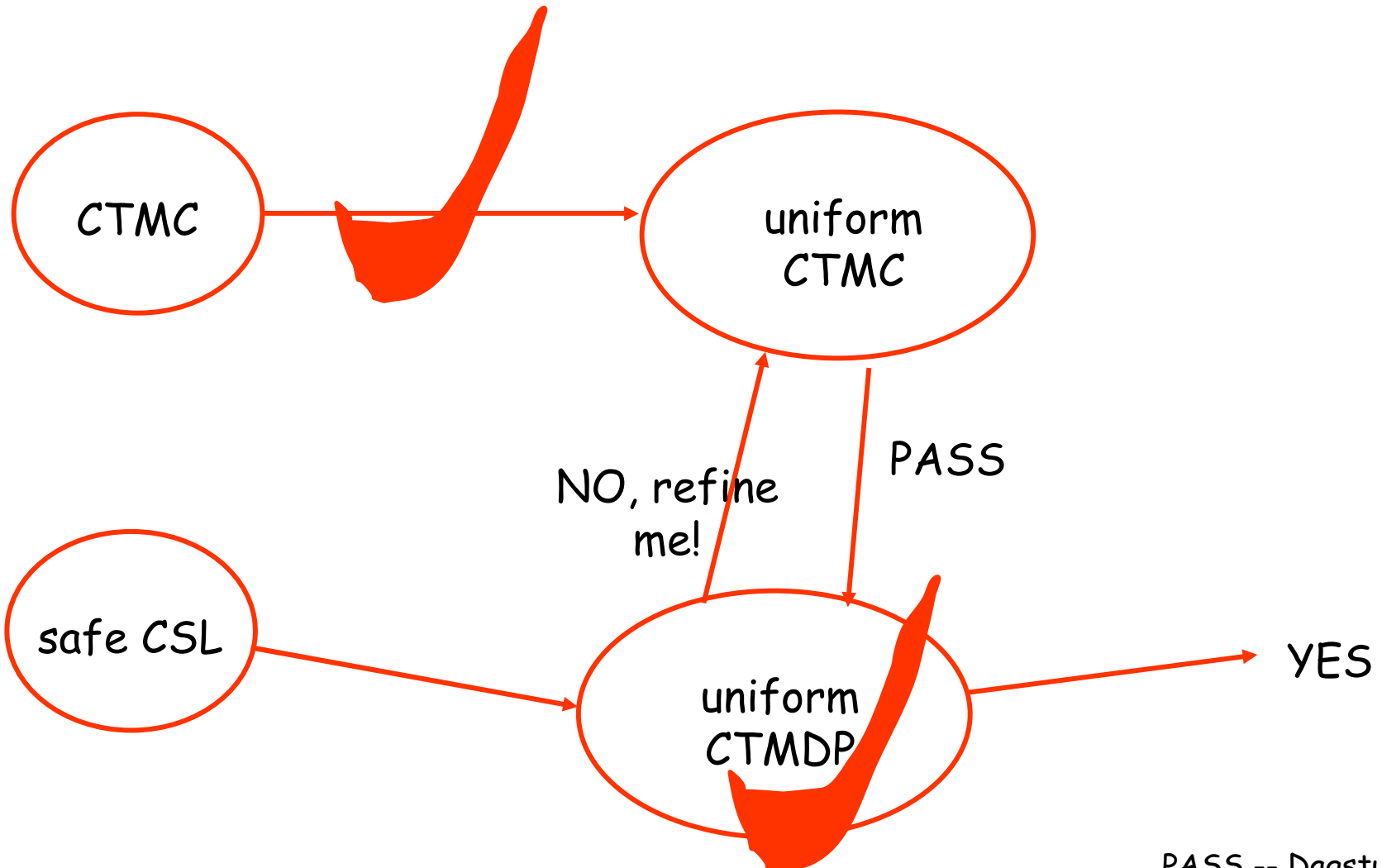
Ongoing works (II)

- Extend to continuous-time models



Ongoing works (II)

- Extend to continuous-time models



Ongoing works (III)

- Extend to live PCTL fragment
 - Combine the game-based abstraction technique [KNP06]

Thank you!